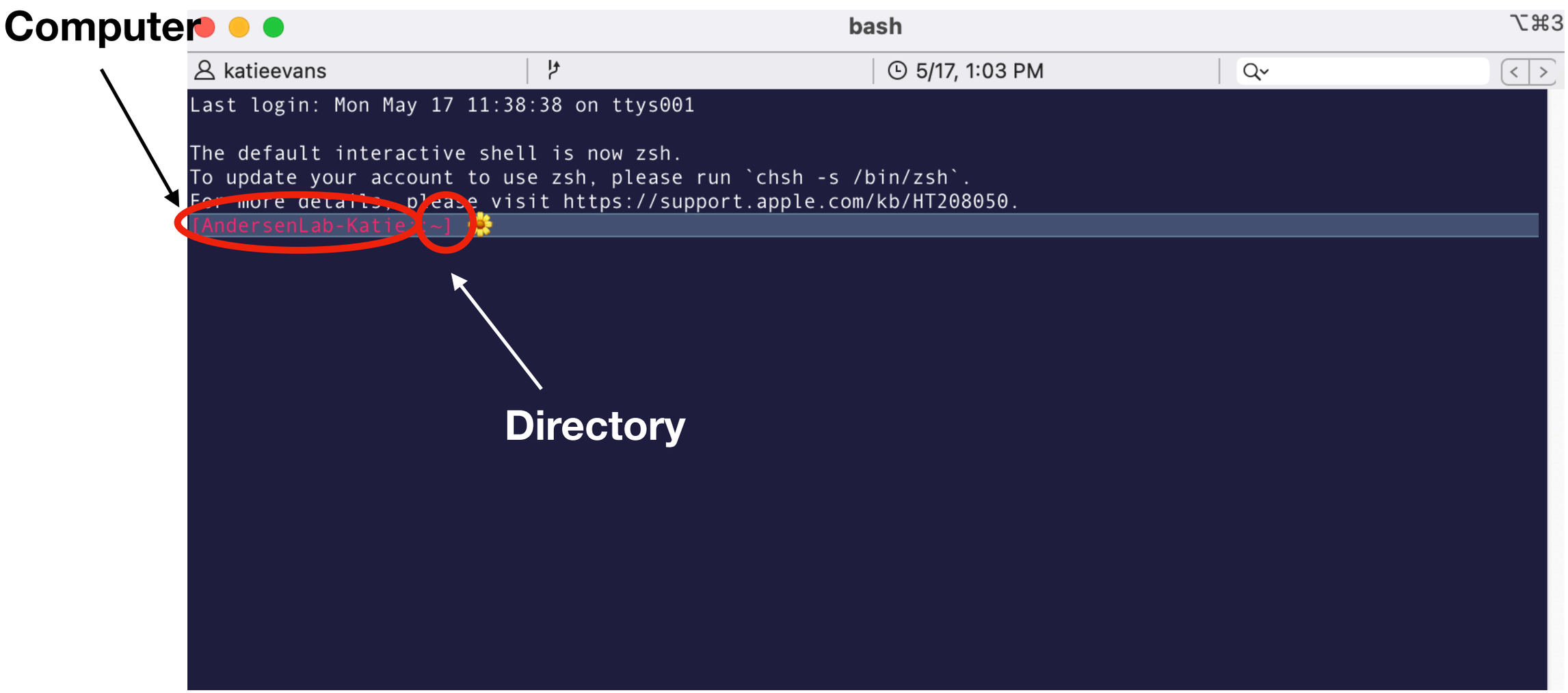
Code Club: Command Line

5.28.21

http://andersenlab.org/dry-guide/latest/bash/

Welcome to terminal!

Your command line!

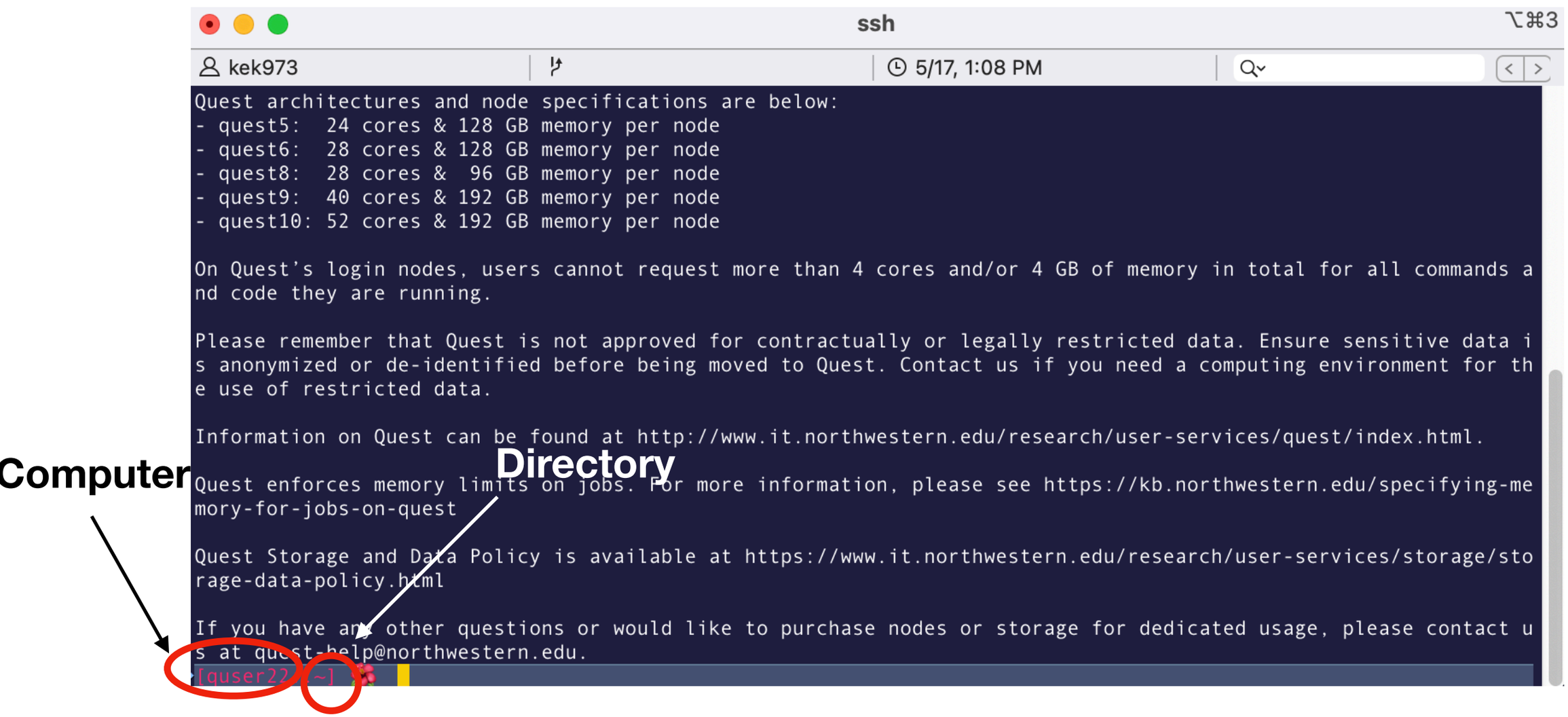


On Mac: Terminal

Other 3rd party apps (like iTerm2 etc.)

Welcome to terminal!

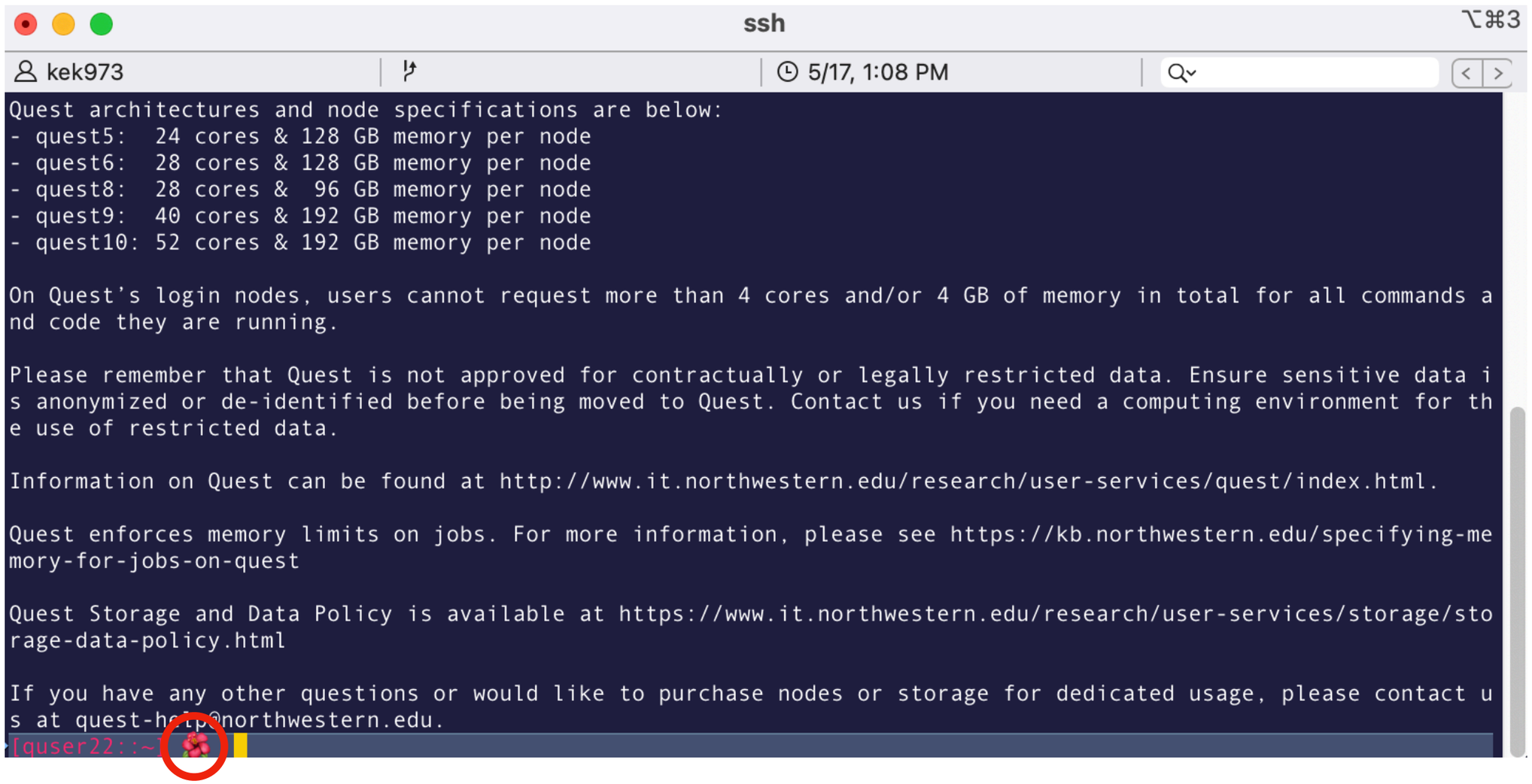
Your command line!



On Mac: Terminal

Other 3rd party apps (like iTerm2 etc.)

Welcome to terminal!




```
vi ~/.bash_profile
```

Cool prompt generator!!!

```
export PS1='\[\e[0;35m\][\h::\W] 🌺 \[\e[m\] '
```

This stuff also determines the colors and what is shown on the prompt

Welcome to terminal!

New terminal tab

`Command + T`

New terminal window

`Command + N`

Show previous command

`Up arrow`

~/.bash_profile

```
alias quest="ssh <netid>@quest.northwestern.edu"
```

`> quest` ⟶ Logs in to quest using ssh ^

Log in to QUEST

```
{ ssh <netid>@quest.it.northwestern.edu
  ssh <netid>@quest.northwestern.edu }
```

(Enter password when prompted)

(Set up ssh key to avoid typing password each time):

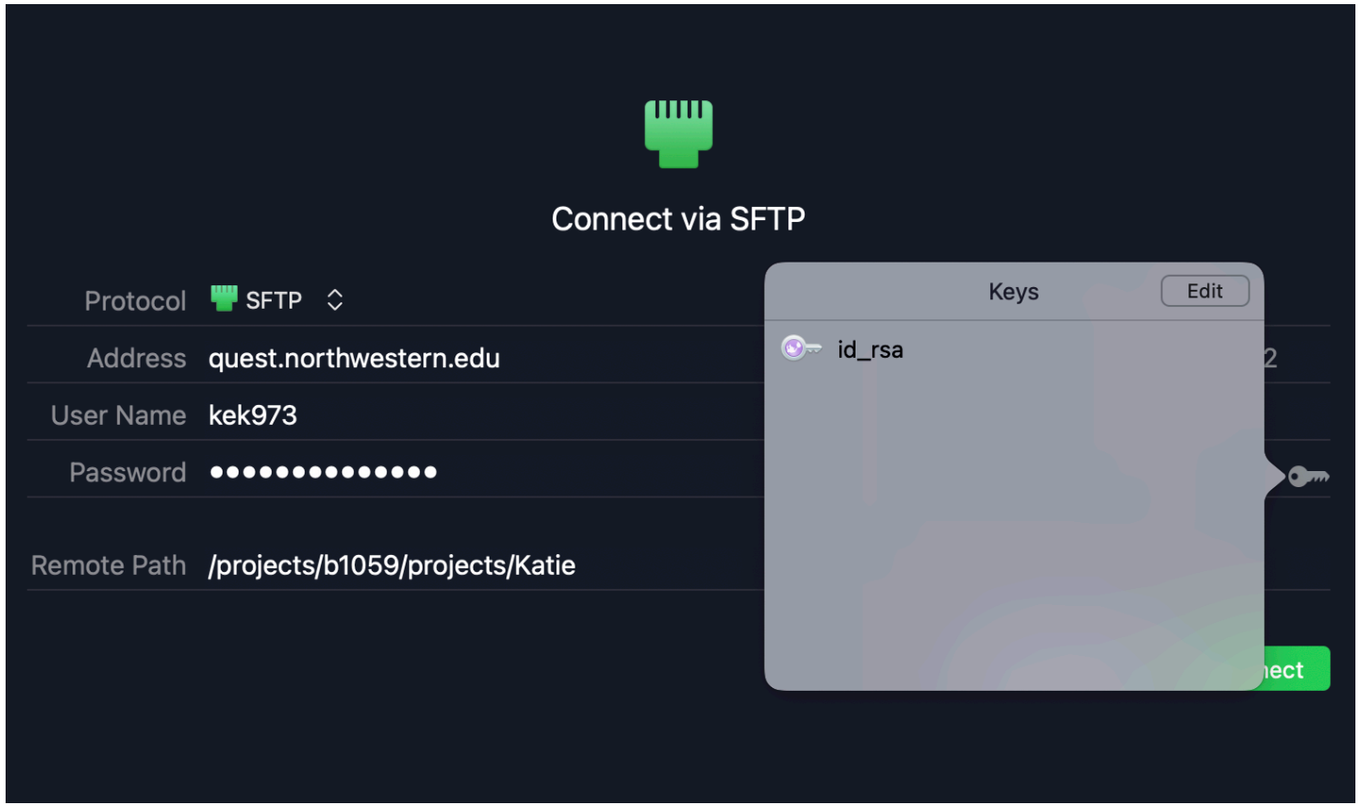
LOCAL

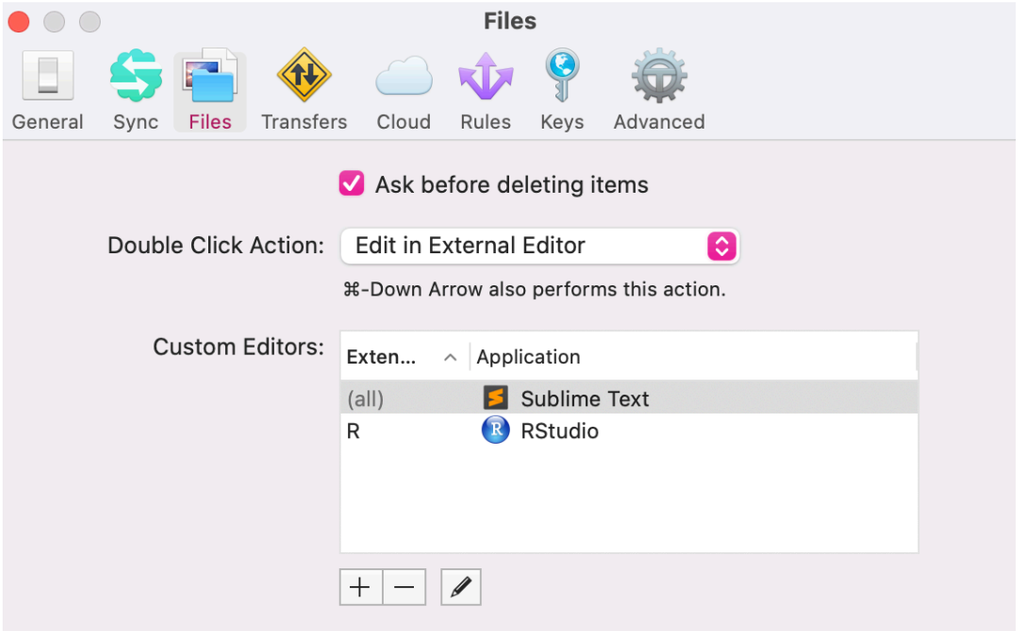
```
cd ~
ssh-keygen
(keep default name, overwrite it if asks, no
password needed, just press ENTER)
cat ~/.ssh/id_rsa.pub
(copy entire output)
```

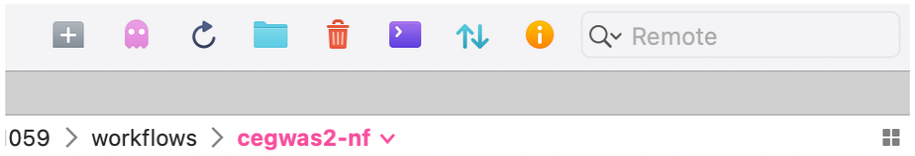
QUEST

```
cd ~/.ssh (or make directory if new)
vi authorized_keys
(paste contents from id_rs.pub, save, exit)
```

Transmit/CyberDuck etc.








```
scp <netid>@quest.northwestern.edu:<path_on_quest>/file.txt .

scp file.txt <netid>@quest.northwestern.edu:<path_on_quest>
```

Terminal alternatives

https://iterm2.com/

https://www.fossmint.com/alternative-terminal-apps-for-macos/

Introduction to the basics

Change directories (hard path)

```
cd ~/Dropbox/AndersenLab/LabFolders/Katie/
```

Don't forget about TAB completion :)

Change directories (relative path)

```
cd ../Clay/
```

Print current directory

```
pwd
```

Create new directory

```
mkdir new_folder
```

Create new file

```
touch new_file.tsv
```

Introduction to the basics

Be careful with this one!! You WILL overwrite if the file already exists. Also try:

```
mv -i new_file.tsv ../Tim
```

Move a file

```
mv new_file.tsv ../Tim/
```

Move (and rename) a file

```
mv new_file.tsv ../Tim/new_file2.tsv
```

Rename a file (without moving)

```
mv new_file.tsv ./new_file2.tsv
```

Copy a file

```
cp new_file.tsv new_file_copy.tsv
```

Open a file

```
open new_file.tsv
```

Introduction to the basics

List files in a directory

`ls` `ls ~/Dropbox/AndersenLab`

List files with more information about the files

`ls -l` `ls -ltr` `ls -lah` (`man ls`) for more options

Delete a file

`rm new_file.tsv`

Delete an <u>empty</u> folder

`rmdir new_folder`

Be careful with this one!! Once deleted, it's gone! Use at your own risk

****Delete a <u>full</u> folder (and all files inside)****

`rm -r new_folder`

`rm -i new_file.tsv`

Introduction to the basics

Create file using redirection

```
ls ~/Dropbox/AndersenLab/LabFolders > people.txt
```

Be careful with this one!! You WILL overwrite previous "people.txt" with this command

Print the contents of a file

```
cat people.txt
```

Print a line of text or string

```
echo "this is a test"
```

Add a new line to an existing file

```
echo "Alyssa" >> people.txt
```

Concatenate (put together) multiple files

```
cat file1.txt file2.txt
```

Introduction to the basics

View contents of a file

```
more file.txt
```

```
less file.txt
```

Scroll up/down
Use **q** to exit

`less` is similar to `more`, but faster because it does not load the entire file at once

Find/search for a file

```
find <directory> -name "file.txt"
```

OR:

```
ls -R * | grep ECA1119
```

Pipe commands together (like %>%)

```
cat file.txt | head
```

Download file from url

```
wget <url>
```

```
wget -bqc <url>
```

```
curl <url>
```

Download in background

Pull out lines with pattern

```
cat file.txt | grep ECA1242
```

Highlight all mention of "ECA1242"

Introduction to the basics

Count words in a file

```
wc file.txt
```

Count lines in a file

```
wc -l file.txt
```

Count number of files in directory

```
ls -ltr | wc -l
```

Print the first few (n) lines of a file

```
head file.txt
```

```
head -n 20 file.txt
```

Print the last few (n) lines of a file

```
tail file.txt
```

Introduction to the basics

Set a variable

```
chr="V";  echo $chr
```

Sort lines in a file

```
head test.tsv | sort -k7
```

Sort by 7th col

Get unique values

```
head test.tsv | awk '{print $7}' | uniq
```

Can only find unique lines next to each other, might need to **sort** first

Intermediate tasks

~ If/else - BASH ~

```
if [[logical]]
then
   [commands]
else
   [commands]
fi
```

~ If/else - R ~

```
if(logical) {
   [commands]
 } else {
   [commands]
 }
```

```
if [[ $sp == “ce” ]]
then
   echo “TRUE”
else
   echo “FALSE”
fi
```

```
if(species == “c_elegans”) {
   print(“TRUE”)
} else {
   print(“FALSE”)
}
```

```
if [[ $sp == “ce” ]]; then echo “TRUE”; else echo “FALSE”; fi
```

Intermediate tasks

~ For loop - BASH ~

```
for item in [LIST]
do
   [commands]
done
```

~ For loop - R ~

```
for(item in [LIST] {
   [commands]
 }
```

```
for base in A C G T
do
   echo “Base: $base”
done
```

```
for(base in c(“A”, “C”, “G”, “T”){
   print(paste(“Base:”, base))
 }
```

```
for base in A C G T; do echo "Base: $base"; done
```

Intermediate tasks

~ For loop - BASH ~

```
for item in [LIST]
do
   [commands]
done
```

```
for file in `ls`
do
   echo $file
done
```

Use` `to implement code in a command

Intermediate tasks

bash_script.sh

```
#!/bin/bash
#SBATCH -J name                # Name of job
#SBATCH -A b1042               # Allocation
#SBATCH -p genomicsguestA      # Queue
#SBATCH -t 24:00:00            # Walltime/duration of the job (hh:mm:ss)
#SBATCH --cpus-per-task=1      # Number of cores (= processors = cpus) for each task
#SBATCH --mem-per-cpu=3G       # Memory per core needed for a job
#SBATCH --array=0-9            # number of parallel jobs to run counting from 0. Make sure to c

# make a list of all the files you want to process
# this script will run a parallel process for each file in this list
name_list=(*.bam)

# take the nth ($SGE_TASK_ID-th) file in name_list
# don't change this line
Input=${name_list[$SLURM_ARRAY_TASK_ID]}

# then do your operation on this file
Output=`echo $Input | sed 's/.bam/.sam/'`
```

sbatch bash_script.sh

squeue -u <netid>

scancel <jobid>

Intermediate: Screen/nohup

Perhaps the most common way to deal with scripts that run for a long time is `screen`. For the most simple case use, type `screen` to open a new screen session and then run your script like normal. Below are some more intermediate commands for taking full advantage of `screen`:

- `screen -S <some_descriptive_name>`: Use this command to name your screen session. Especially useful if you have several scren sessions running and/or want to get back to this particular one later.
- `Ctrl+a` + `Ctrl+d` to detach from the current screen session
- `exit` to end the current screen session
- `screen -ls` lists the IDs of all screen sessions currently running
- `screen -r <screen_id>`: Use this command to resume a particular screen session. If you only have one session running you can simply use `screen -r`

Important

When using `screen` on QUEST, take note that screen sessions are only visible/available when you are logged on to the particular node it was created on. You can jump between nodes by simply typing ssh and the login node you want (*e.g.* `ssh quser22`).

Intermediate: Screen/nohup

Another way to avoid **SIGHUP** errors is to use `nohup`. `nohup` is very simple to use, simply type `nohup` before your normal command and that's it!

```
nohup nextflow run main.nf
```

Running a command with `nohup` will look a little different than usual because it will not print anything to the console. Instead, it prints all console outputs into a file in the current directory called `nohup.out`. You can redirect the output to another filename using:

```
nohup nextflow run main.nf > cegwas_{date}_output.txt
```

When you exit this window and open a new session, you can always look at the contents of the output file using `cat nohup.out` to see the progress.

Note

You can also run `nohup` in the background to continue using the same window for other processes by running `nohup nextflow run main.nf &`.

Intermediate: Software/conda

Modules

- module avail # what Quest has
- module list # what I have loaded
- module add/load # load a module
- module purge # remove all modules

Conda

- module load python/anaconda
- conda create -n name_env # create environment
- conda activate name_env # go inside the env

  source activate name_env # if the line above doesn't work
- conda install -c bioconda nextflow=0.20.0

  conda install nextflow # if .condarc is set up
- conda deactivate # go inside the env

  source deactivate # if the line above doesn't work

~ Dan Lu ~

Intermediate: Software/conda

	benefits	limitations
Manual install	- Full control by users	- May fail - Less easy to turn on and off
Modules	- Easiest to use	- Not managed by users - Don't know what's in there or what changes
Conda	- Easy to use - Easy to share and reproduce - Bundles of packages	- Still sees the other packages
Docker/ Singularity	- Fully isolated environment - Highly portable	- Takes a bit more work to set up

~ Dan Lu ~

Advanced: bcftools

- **annotate** .. edit VCF files, add or remove annotations
- **call** .. SNP/indel calling (former "view")
- **cnv** .. Copy Number Variation caller
- **concat** .. concatenate VCF/BCF files from the same set of samples
- **consensus** .. create consensus sequence by applying VCF variants
- **convert** .. convert VCF/BCF to other formats and back
- **csq** .. haplotype aware consequence caller
- **filter** .. filter VCF/BCF files using fixed thresholds
- **gtcheck** .. check sample concordance, detect sample swaps and contamination
- **index** .. index VCF/BCF
- **isec** .. intersections of VCF/BCF files
- **merge** .. merge VCF/BCF files files from non-overlapping sample sets
- **mpileup** .. multi-way pileup producing genotype likelihoods
- **norm** .. normalize indels
- **plugin** .. run user-defined plugin
- **polysomy** .. detect contaminations and whole-chromosome aberrations
- **query** .. transform VCF/BCF into user-defined formats
- **reheader** .. modify VCF/BCF header, change sample names
- **roh** .. identify runs of homo/auto-zygosity
- **sort** .. sort VCF/BCF files
- **stats** .. produce VCF/BCF stats (former vcfcheck)
- **view** .. subset, filter and convert VCF and BCF files

Advanced: bcftools

Do you have bcftools??

```
module load bcftools
```

View VCF

```
bcftools view <vcf>
```

View VCF only header (-h) or without header (-H)

```
bcftools view -h <vcf>
```

```
bcftools view -H <vcf>
```

Subset vcf for only 3 samples

```
bcftools view -s CB4856,XZ1516 <vcf>
```

Subset vcf for a region of interest

```
bcftools view -r III:1-800000 <vcf>
```

Advanced: bcftools

Print out contents of vcf in a specific format

```
bcftools query -f '%CHROM\t%POS\t%REF\t%ALT[\t%SAMPLE=%GT]\n' <vcf> > out.tsv
```

Print list of samples in vcf

```
bcftools query -l <vcf>
```

Keep rows that include a tag (i.e. filter?)

```
bcftools query -i GT==“alt” <vcf>
```

Remove rows that include a tag

```
Bcftools query -e GT==“ref” <vcf>
```

Bcftools manual

Bcftools cheat sheet

Advanced: awk

- ‘awk’ is a utility/language designed for **data extraction**
- Often used in combination with **sed**
- Reads one line at a time
- syntax: **`awk ‘condition {action}’`**
- Awk guide, **examples**

Advanced: awk

$1 = 1st column/field
$0 = all cols
NF = number of fields
$NF = last col
$(NF-1) = second to last

Print only columns 1 and 3 using stdin

```
awk ‘{print $1,$3}'
```

Print elements from column 2 that match a pattern using stdin (filter)

```
awk ‘/HIGH/ {print $6}’
```

regexp is specified within //

Create new column

```
echo 'foo,bar,123,baz' | awk -F, -v OFS=, '{$5=42} 1'
```

```
awk 'BEGIN { FS = OFS = "\t" } { $(NF+1) = 1; print $0 }' infile
```

Prints every record between 2405089 and 3729798 (2nd col)

```
awk ‘$2 == 2405089, $2 == 3729798’ test.tsv
```

Advanced: awk

Use a different delimiter

```
echo ‘foo:123:bar:789’ | awk -F’:' ‘{print $2}’
```

Split with <u>regular expressions</u>

```
echo ‘Sample123string54with908numbers’ | awk -F’[0-9]+’ ‘{print $2}’
```

Change csv to tsv

```
echo 'foo,123,bar,789' | awk -F',' -v OFS='\t' '{$1=$1; print $0}'
```

```
echo 'foo,123,bar,789' | awk ‘{gsub(“,”, “\t”)} 1’
```

Print every line except header

```
awk ‘NR>1’ inputfile.tsv
```

Advanced: awk

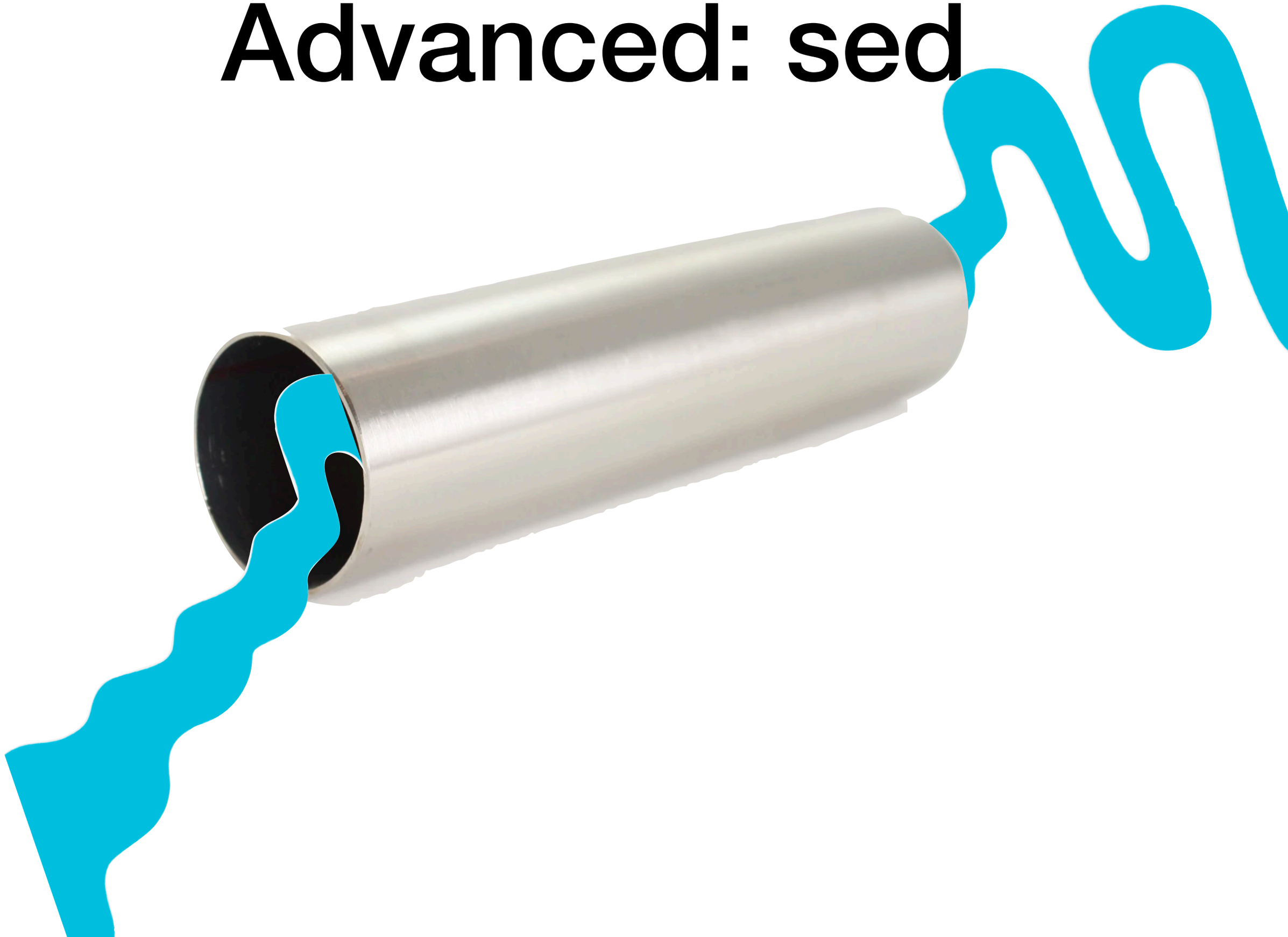
Print header and lines (columns 1 and 2) where position < 3000

```
awk ‘NR==1 || $2<3000 {print $1,$2}’ inputfile.tsv
```

```
Awk ‘{
    if(NR==1 || $2<3000){
        print $1,$2
    }
}’ inputfile.tsv
```

SO SO SO SO MUCH MORE (I know nothing…)

Advanced: sed



Advanced: sed

Doesn't need quotes, but good practice

Essential command: s for substitution

```
sed s/pattern/replacement/ input > output
```

```
echo ‘C. elegans is amazing!' | sed ’s/elegans/briggsae/‘
```

Think about the substitutions you are asking it to make…

```
echo ‘Sunday funday’ | sed ’s/day/night/‘
```

Default behavior is to replace once per line… to change that, use ‘g'

```
echo ‘Sunday funday' | sed ’s/day/night/g’
```

https://forms.gle/S1vWAw3kXynAZZzV6